



EDINBURGH NAPIER UNIVERSITY

**SET08101 Web Tech**

---

## **Lab 6 - Beginning Node.JS**

---

Dr Simon Wells

---

# 1 Aims

At the end of the practical portion of this topic you will:

- 

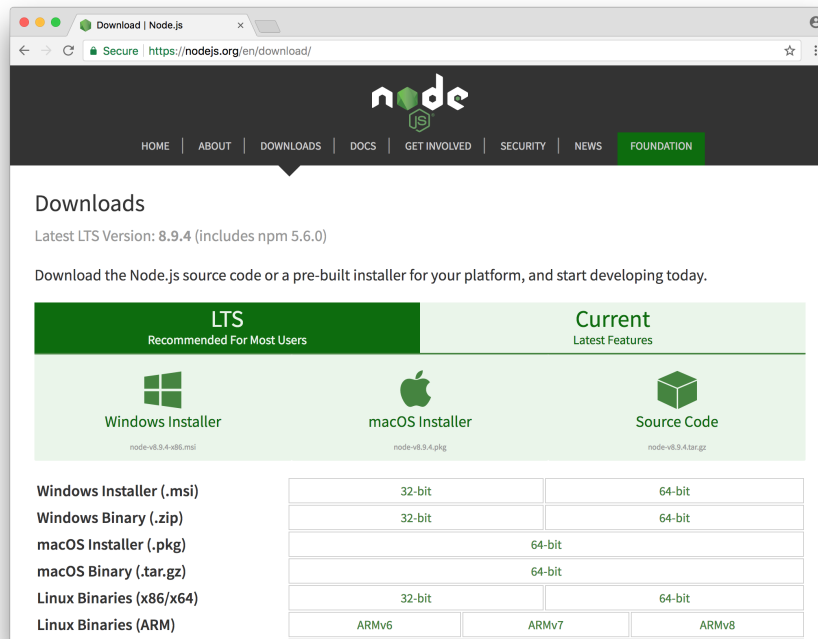
## 1.1 Introduction

We've concentrated so far on JavaScript in the browser or *on the client side*. Now let's look at using JS on the server as well. We'll start by running the Node console and using it as a place to type in JS code. This is quite a powerful little tool that we can use to work with JS as just another programming language rather than hosting it in a browser linked to HTML files. We'll then look at using Node as a HTTP server, to host

## 2 Activities

### 2.1 Getting Node.JS

Download the Node.JS Windows Binary distribution (zip file) from the node.js website: <https://nodejs.org/en/download/><sup>1</sup>. Don't use the MSI download if you're on a lab machine because this might need administrative privileges to install.



Once you've downloaded the zip, extract it to somewhere suitable such as a folder on your H: drive or on USB stick. Don't just double click zip file but instead ensure that you've properly extracted the contents of the zip file. If you have trouble extracting the contents of the zip file then use a decent zip archive tool like 7zip<sup>2</sup> instead.

Once you've extracted the Node files we should have a folder containing some useful tools. The most important here is node.exe which we can use to run the Node program, but we'll get to that in the next section...

If you're on a Mac then use homebrew to install Node, e.g.

```
$ brew install node
```

---

<sup>1</sup>Or directly download this zip: <https://nodejs.org/dist/v8.9.4/node-v8.9.4-win-x86.zip>

<sup>2</sup>7zip portable download: <https://portableapps.com/apps/utilities/7-zip-portable>

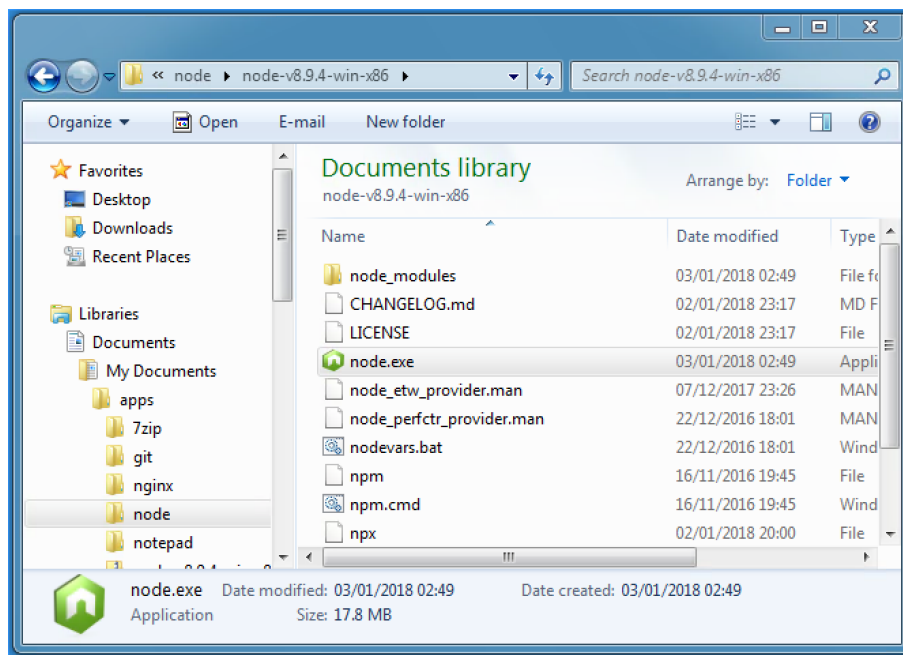
If you're on Linux then use APT, or you're distro's package manager to install Node, e.g.

```
$ apt-get install node
```

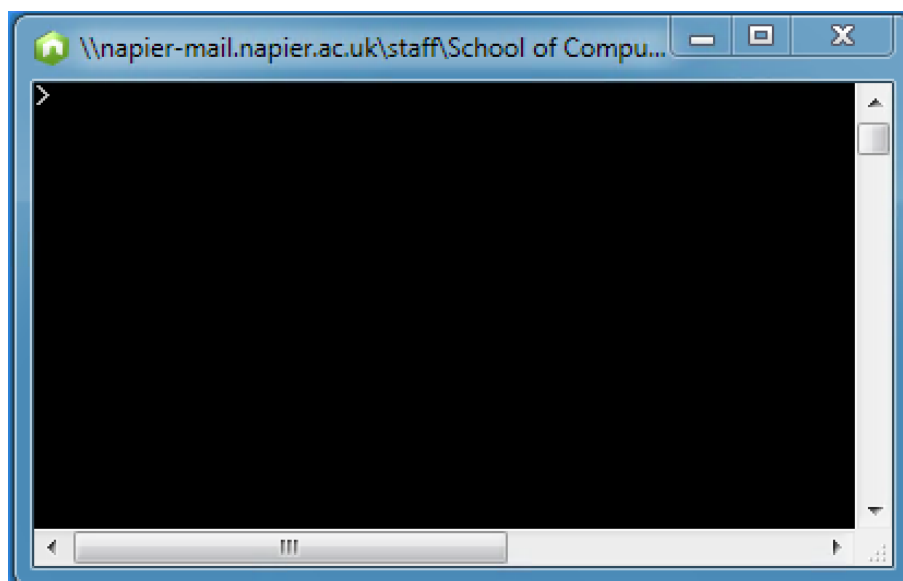
## 2.2 Node.JS Console

We can run node and use JS as just another programming language rather than as a *web* programming language. That is we can write some JS and execute it without having to have an HTML file to host our code and load it into the browser. In fact, we don't even need the browser at this point, although we will quickly return to using the browser as this is a web technologies module after all.

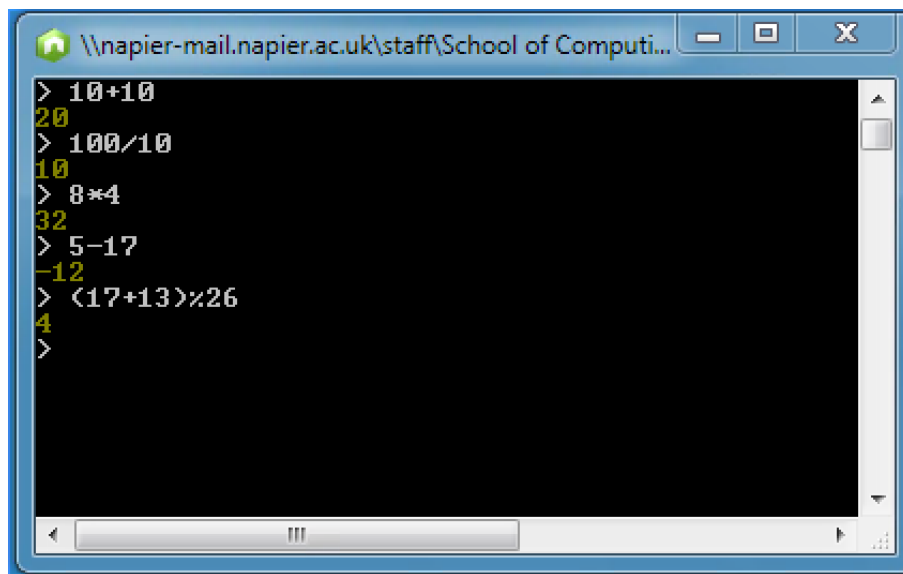
Node comes complete with a console. A place where we can type JS code and see the output immediately. Such an environment is known as REPL, a Read-Evaluate-Print-Loop environment; it reads a command, works out what to do, prints the result, then loops back to the start and reads the next thing. We can get a node REPL by double clicking the node.exe program:



If all has gone well we should see a window like this:

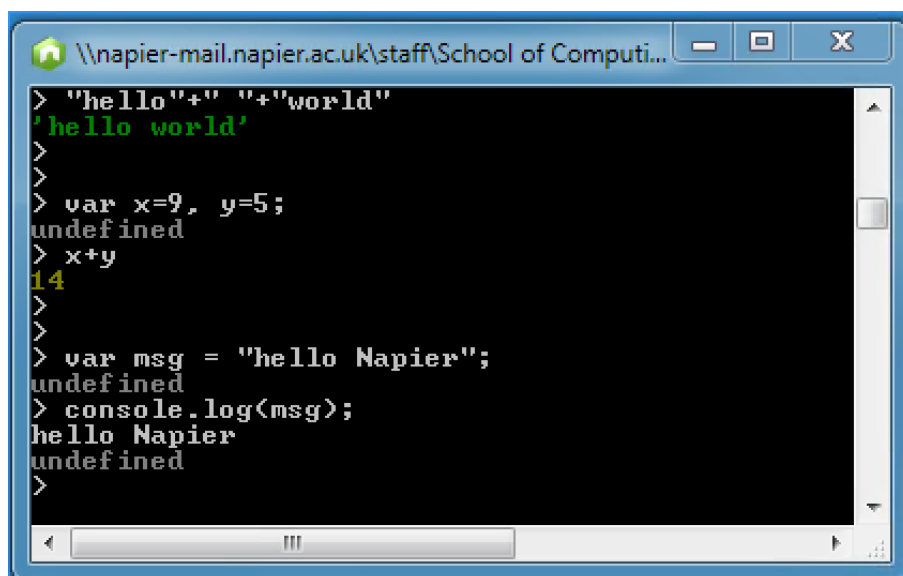


This is a place that we can type JS code into, why not give it a go? Let's start with some basic arithmetic:



```
> 10+10
20
> 100/10
10
> 8*4
32
> 5-17
-12
> (17+13)*26
4
>
```

We can use `ctrl+l` to clear the screen at any time and `ctrl+c` twice to exit<sup>3</sup> from the REPL. As well as arithmetic we can also do standard JS programming, like the following:

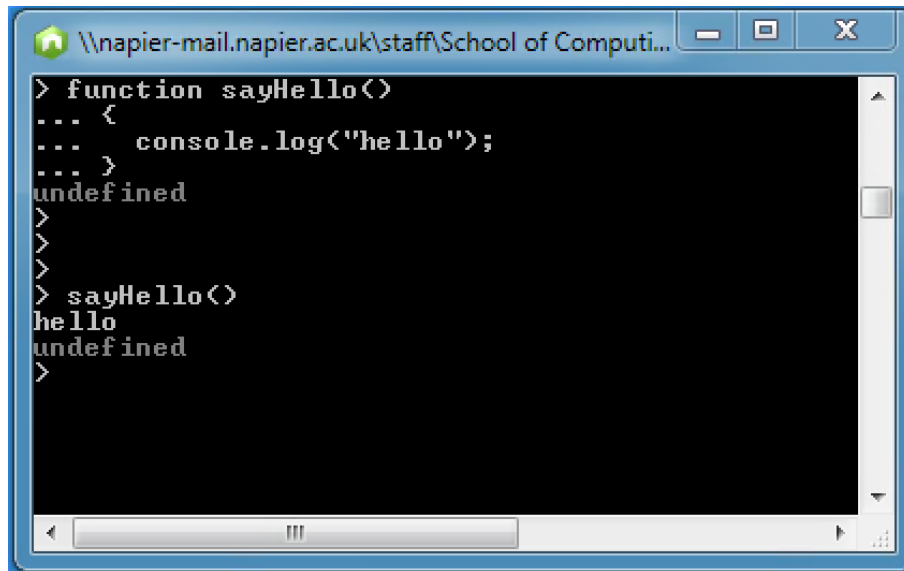


```
> "hello"+" "+"world"
'hello world'
>
> var x=9, y=5;
undefined
> x+y
14
>
> var msg = "hello Napier";
undefined
> console.log(msg);
hello Napier
undefined
>
```

We can also do multiline input by just pressing return wherever we need to. For example, to type in a function:

---

<sup>3</sup>We can also exit the REPL by typing `.exit`


 A screenshot of a Node.js REPL window. The window title is "\\napier-mail.napier.ac.uk\\staff\\School of Computi...". The prompt is ">". The user has entered a function definition:
 

```
> function sayHello()
... <
...   console.log("hello");
... >
undefined
>
>
> sayHello()
hello
undefined
>
```

 The ellipses (...) indicate that the REPL is in continuity mode, allowing the user to write the next line as a continuation of the previous code.

This is called continuity mode, because when we press return we are allowed by the REPL to write the next line as a continuation of our previous code. When in continuity mode we will see the ‘...’ as our prompt instead of ‘>’. You can leave continuity mode by typing ‘.break’ if needed.

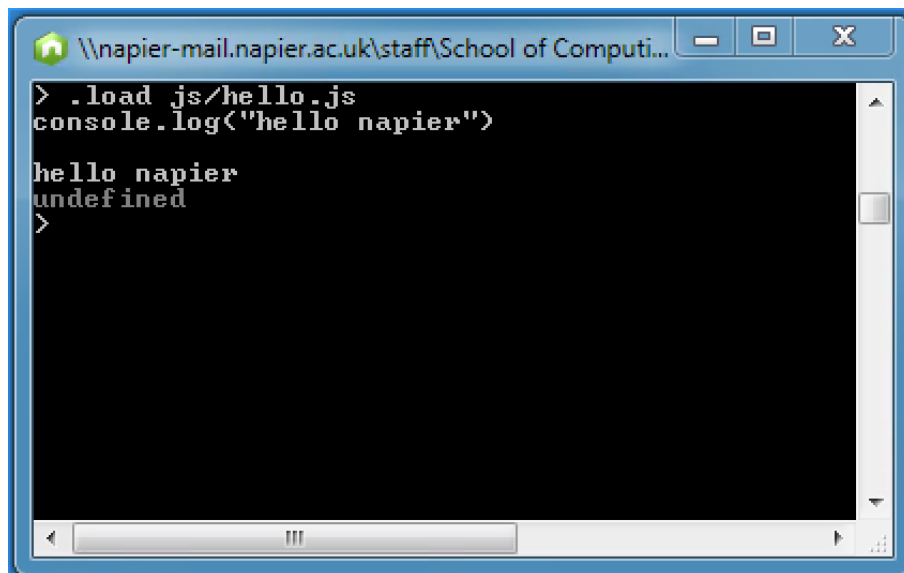
Node has a bunch of useful commands that you can explore, for example, *.help* which gives you information about some of the basic REPL commands. The tab key will display all the available commands. You can also use the up and down arrow keys to cycle through previous commands and lines of code that you’ve already typed in to stop your have to re-type things. The *.save* command will enable you to save your current REPL session, and *.load* will load the specified file into the Node REPL session. Ctrl+C (once) will terminate the current command and ctrl+c (twice) will exit the REPL.

## 2.3 Node.JS as a JavaScript execution environment

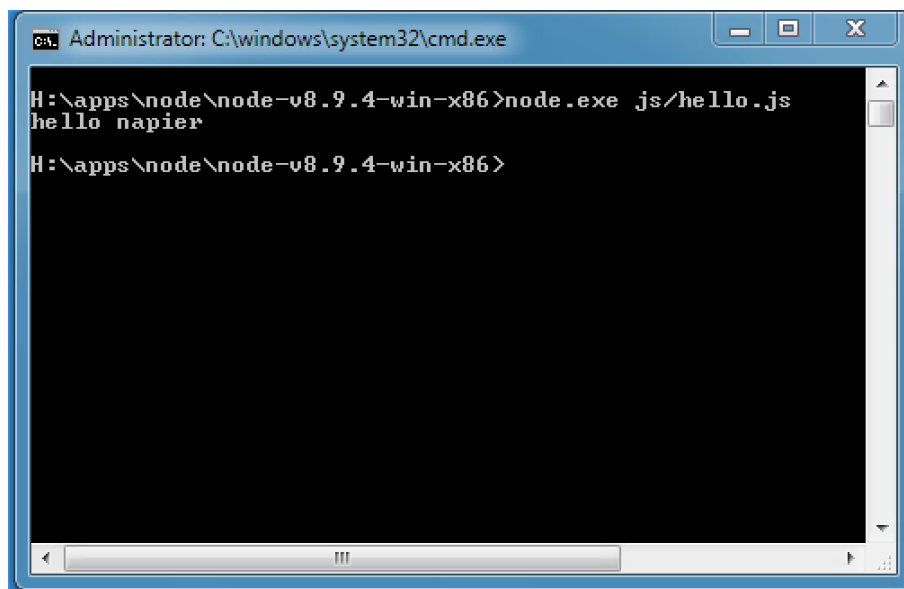
We can load and run JS written in external files in two ways. Firstly by loading them within the REPL and secondly by passing the JS file to be executed to node.exe when we start it. Let’s try both out. For convenience let’s first create a simple JS file called “hello.js” and add the following code to it using a preferred editor:

```
1 console.log('Hello Napier');
```

I saved this to a subfolder of the node folder called js so that I could easily access examples. You can save your JS files wherever you like but you’ll have to specify where they are to Node. Now you can start node and use the ‘.load’ command to load and execute your external JS file, e.g.



Alternatively, instead of clicking node.exe we can start Node from the command line (cmd.exe)<sup>4</sup> and tell it which JS file to load as an argument to the executable. For example:



Using these techniques we can pretty much use node.exe as our own little programming platform. But there's more, so much more we can do with Node. We've barely scratched the surface...

## 2.4 Node.JS as a hosting platform

In addition to hosting your server-side JavaScript, node can also host your regular HTML files. This can be a useful way to test out your site locally or to simplify your deployment. For example, to avoid having to install a full additional web-server if you already have node installed, or else to serve up a small number of static pages in addition to a web-app generation from JS.

We'll need to install the libraries that Node will use. We'll do this by opening the Windows command prompt, then navigating to the location of node.exe, but this time we're going to use the "npm" command:

```
$ npm install connect serve-static
```

<sup>4</sup>Go to the Start menu and type "cmd.exe" into the "Search programs and files" box.

The Node Package Manager (NPM)<sup>5</sup> is a way to package up and distribute JS code for use with Node. We just installed two packages, one called *connect* and the other called *serve-static* which we'll use to serve up our existin HTML.

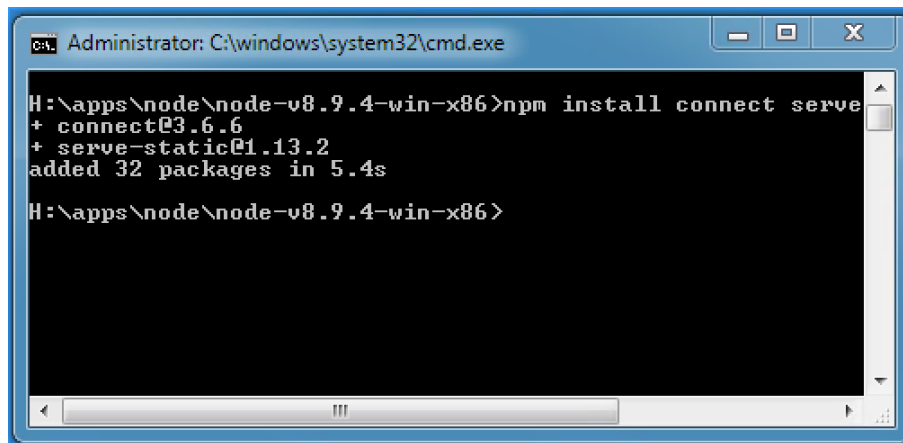
Now we need to create a small JS file, let's call it 'server.js', to tell node how to serve our HTML:

```
1 var connect = require('connect');
2 var serveStatic = require('serve-static');
3 connect().use(serveStatic(__dirname)).listen(8080, function(){
4   console.log('Server running on 8080...');
5 });
```

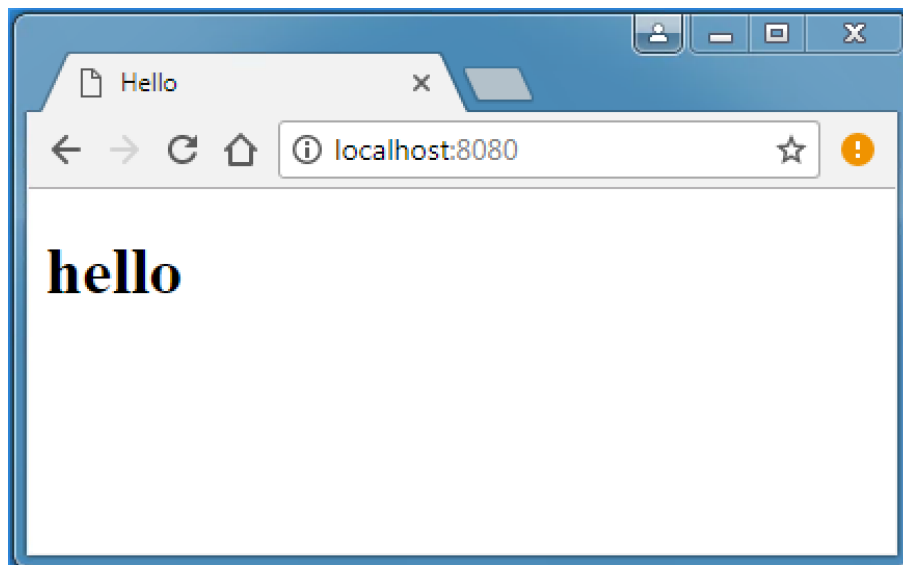
Now we also need a small index.html to serve up, e.g.

```
1 <!DOCTYPE html>
2 <html>
3   <head><title>Hello</title></head>
4   <body>
5     <h1>hello</h1>
6   </body>
7 </html>
```

Put both index.html and server.js into the js folder then invoke node.js to run server.js, e.g.



We should now be able to navigate to either <http://localhost:8080> or <http://127.0.0.1:8080> in our web browser and see our HTML file:



<sup>5</sup>It's well worth visiting the NPM site <https://www.npmjs.com/> and doing some background reading on this technology. There are thousands of ready made packages available through the NPM.

---

### 3 Next

There are a whole bunch of Node.JS libraries that we can build on so that Node can act as a complete server-side web language. These libraries will help us to construct our own web-apps and APIs so that we can generate our user interface on-the-fly based upon what our user wants. This is how many highly interactive websites function and developing skills in this area will enable us to begin the process of going beyond the (mostly-) static websites that we've developed so far.